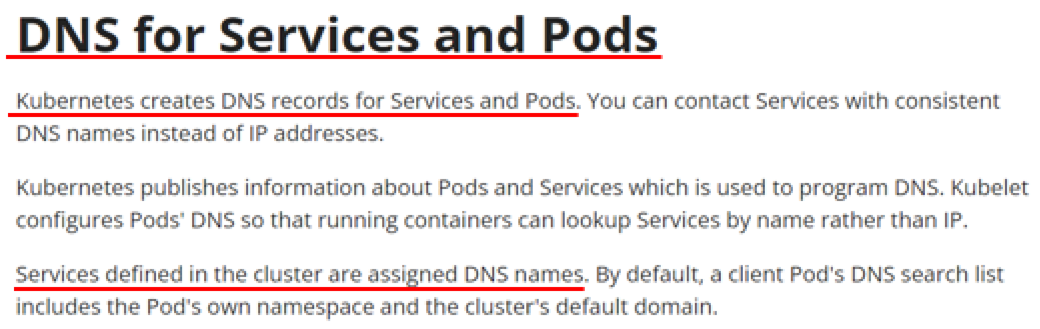
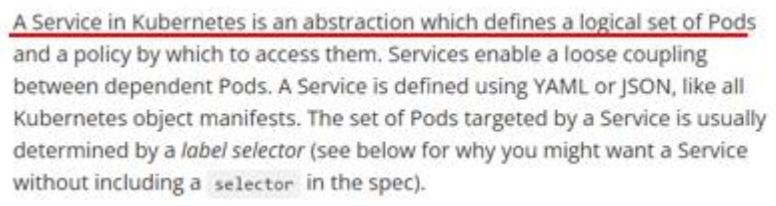
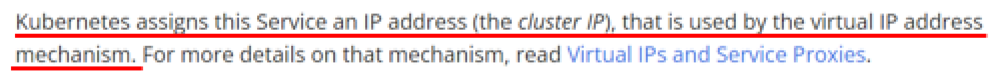
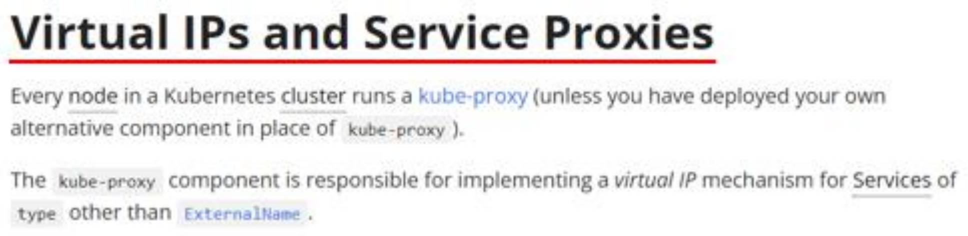
# Exhibit 9

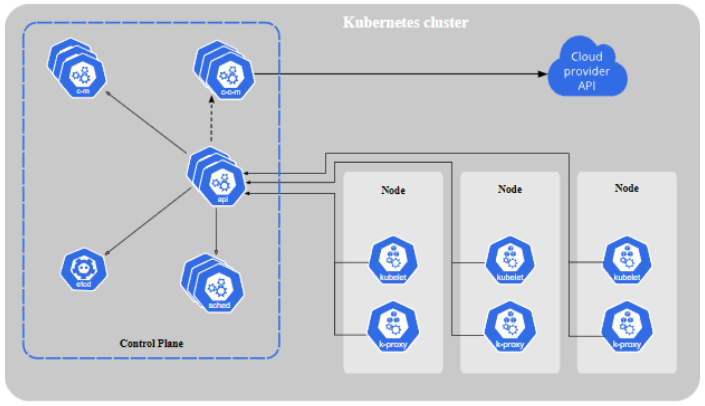
# Exhibit 9 to Complaint
**Intellectual Ventures I LLC and Intellectual Ventures II LLC**

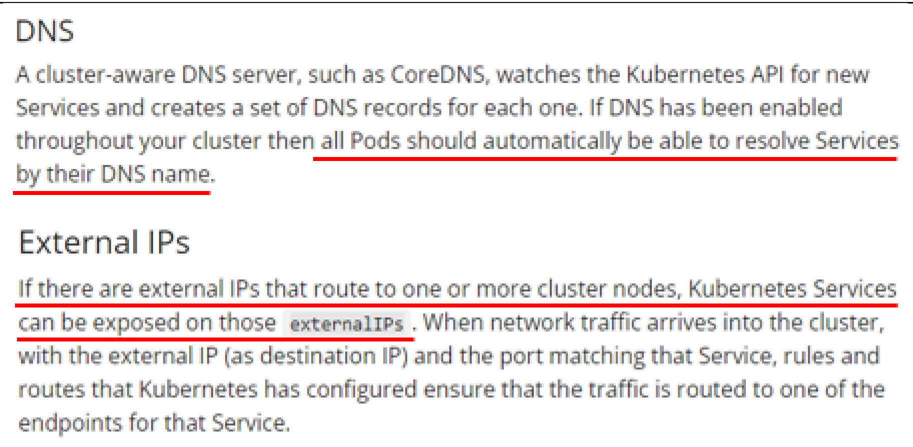
## Example Comerica Count IV Systems and Services
## U.S. Patent No. 7,949,785 ("the '785 Patent")

The Accused Systems and Services include, without limitation, Comerica's systems that utilize Apache Spark ("Spark"); all past, current and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current and future Comerica's systems and services that have the same or substantially similar features as the specifically identified systems and services ("Example Comerica Count IV Systems and Services").

| **U.S. Patent No. 7,949,785** | |
|---|---|
| **Example Claim 30** | **Accused Instrumentalities: Example Comerica Count IV Systems and Services** |
| 30. A virtual network manager, comprising: | Upon information and belief, the Comerica's systems include "a virtual network manager." The virtual network manager is the Kubernetes functionality related to DNS for Services and Pods.<br><br>**DNS for Services and Pods**<br>Kubernetes creates DNS records for Services and Pods. You can contact Services with consistent DNS names instead of IP addresses.<br>Kubernetes publishes information about Pods and Services which is used to program DNS. Kubelet configures Pods' DNS so that running containers can lookup Services by name rather than IP.<br>Services defined in the cluster are assigned DNS names. By default, a client Pod's DNS search list includes the Pod's own namespace and the cluster's default domain.<br>*See* https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/#pod-s-dns-config (last accessed on November 11, 2023).<br><br>A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them. Services enable a loose coupling between dependent Pods. A Service is defined using YAML or JSON, like all Kubernetes object manifests. The set of Pods targeted by a Service is usually determined by a *label selector* (see below for why you might want a Service without including a `selector` in the spec).<br>*See* https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/ (last accessed on November 11, 2023). |

| | |
|---|---|
| |  Kubernetes assigns this Service an IP address (the *cluster IP*), that is used by the virtual IP address mechanism. For more details on that mechanism, read Virtual IPs and Service Proxies. <br><br> *See* https://kubernetes.io/docs/concepts/services-networking/service/ (last accessed on November 11, 2023). <br><br>  Virtual IPs and Service Proxies <br> Every node in a Kubernetes cluster runs a kube-proxy (unless you have deployed your own alternative component in place of `kube-proxy`). <br> The `kube-proxy` component is responsible for implementing a *virtual IP* mechanism for Services of `type` other than `ExternalName`. <br><br> *See* https://kubernetes.io/docs/reference/networking/virtual-ips/ (last accessed on November 11, 2023). <br><br> Comerica's systems include Kubernetes functionality. *See, e.g.*, https://careers.comerica.com/job/19419241/principal-engineer-cloud-automation-governance-and-engineering-cage-dallas-or-detroit-metro-auburn-hills-mi/ (last accessed on November 13, 2023). |
| a network interface configured for data communication via a virtual network that is defined by a domain name having an associated public network address; | Upon information and belief, the Comerica's systems utilize Kubernetes and include "a network interface configured for data communication via a virtual network that is defined by a domain name having an associated public network address." For example, Kubernetes DNS for Services and PODS is implemented in part through a network interface called kube-proxy, which maintains network rules that allow communication to pods from network sessions. Kube-proxy communications can relate to network sessions inside or outside of the cluster (the virtual network) in which the services are defined. The services defined in the cluster are assigned DNS names having an associated public network address. |

| | |
|---|---|
| | **Kubernetes Components**<br> Kubernetes cluster<br>Cloud provider API<br>c-m<br>c-c-m<br>api<br>etcd<br>sched<br>Control Plane<br>Node<br>Node<br>Node<br>kubelet<br>k-proxy<br>kube-proxy<br>kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.<br>kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.<br><br>*See* https://kubernetes.io/docs/concepts/overview/components/ (last accessed on November 11, 2023). |



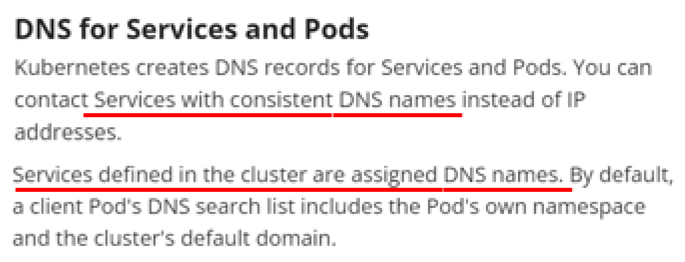
DNS

A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name.

External IPs

If there are external IPs that route to one or more cluster nodes, Kubernetes Services can be exposed on those `externalIPs` . When network traffic arrives into the cluster, with the external IP (as destination IP) and the port matching that Service, rules and routes that Kubernetes has configured ensure that the traffic is routed to one of the endpoints for that Service.

*See* https://kubernetes.io/docs/concepts/services-networking/service/ (last accessed on November 11, 2023).

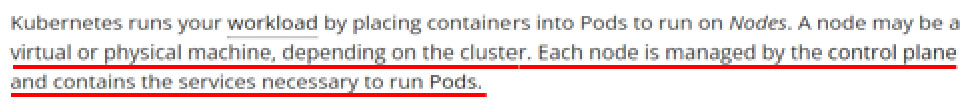
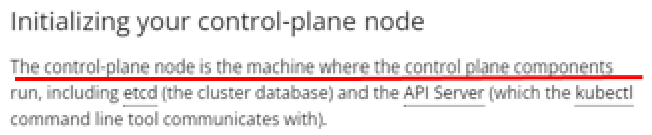
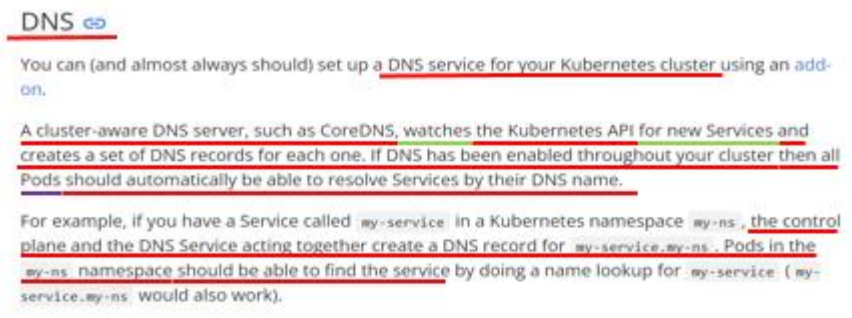


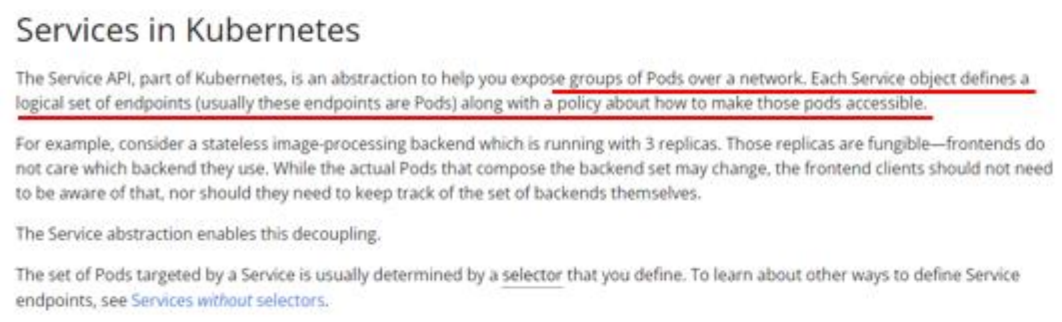
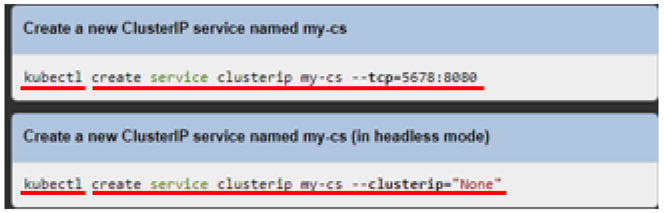
**DNS for Services and Pods**

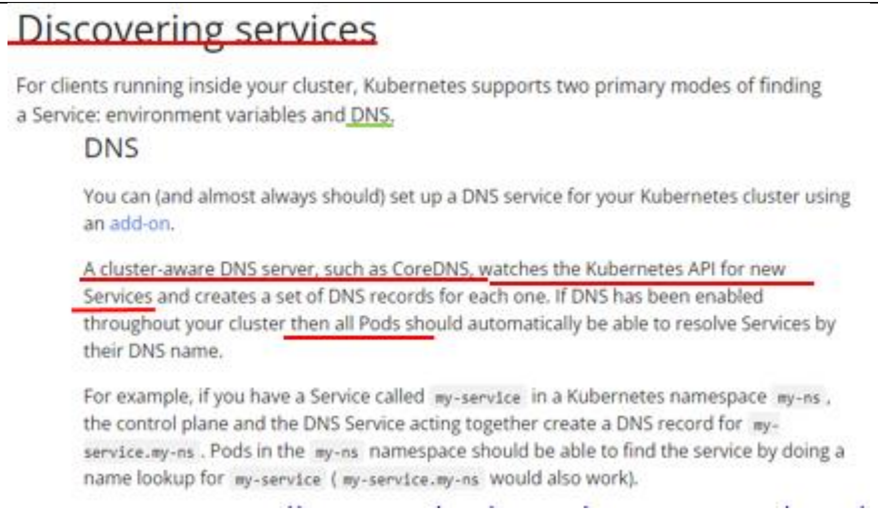
Kubernetes creates DNS records for Services and Pods. You can contact Services with consistent DNS names instead of IP addresses.

Services defined in the cluster are assigned DNS names. By default, a client Pod's DNS search list includes the Pod's own namespace and the cluster's default domain.

*See* https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/#pod-s-dns-config (last accessed on November 11, 2023).

| | |
|---|---|
| a memory and a processor to implement a register module configured to register devices in a virtual network | Upon information and belief, the Comerica's systems utilize Kubernetes and includes "a memory and a processor to implement a register module configured to register devices in a virtual network." A service runs on a set of pods (devices). The Kubernetes DNS Service module (register module) watches the Kubernetes API for incoming new services and creates a set of DNS records for each of the pods associated with the service (register devices).<br><br>Kubernetes runs your workload by placing containers into Pods to run on *Nodes*. A node may be a virtual or physical machine, depending on the cluster. Each node is managed by the control plane and contains the services necessary to run Pods.<br>*See* https://kubernetes.io/docs/concepts/architecture/nodes (last accessed on November 11, 2023).<br><br>Initializing your control-plane node<br>The control-plane node is the machine where the control plane components run, including etcd (the cluster database) and the API Server (which the kubectl command line tool communicates with).<br>*See* https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/ (last accessed on November 11, 2023).<br><br>DNS<br>You can (and almost always should) set up a DNS service for your Kubernetes cluster using an add-on.<br>A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name.<br>For example, if you have a Service called `my-service` in a Kubernetes namespace `my-ns`, the control plane and the DNS Service acting together create a DNS record for `my-service.my-ns`. Pods in the `my-ns` namespace should be able to find the service by doing a name lookup for `my-service` (`my-service.my-ns` would also work).<br>*See* https://kubernetes.io/docs/concepts/services-networking/service (last accessed on November 11, 2023). |

| | |
|---|---|
| |  Services in Kubernetes<br>The Service API, part of Kubernetes, is an abstraction to help you expose groups of Pods over a network. Each Service object defines a logical set of endpoints (usually these endpoints are Pods) along with a policy about how to make those pods accessible.<br>For example, consider a stateless image-processing backend which is running with 3 replicas. Those replicas are fungible—frontends do not care which backend they use. While the actual Pods that compose the backend set may change, the frontend clients should not need to be aware of that, nor should they need to keep track of the set of backends themselves.<br>The Service abstraction enables this decoupling.<br>The set of Pods targeted by a Service is usually determined by a selector that you define. To learn about other ways to define Service endpoints, see Services *without* selectors.<br>*See* https://kubernetes.io/docs/reference/kubectl/cheatsheet/ (last accessed on November 11, 2023). |
| the register module further configured to:<br>receive a registration request from an agent associated with a device; | Upon information and belief, the Comerica's systems include "the register module further configured to receive a registration request rom an agent associated with a device." A service runs on a set of pods. The Kubernetes DNS Service module (register module) watches the Kubernetes API for incoming new services (registration requests) being made via tools like kubctl (agent associated with a device).<br> Create a new ClusterIP service named my-cs<br>`kubectl create service clusterip my-cs --tcp=5678:8080`<br>Create a new ClusterIP service named my-cs (in headless mode)<br>`kubectl create service clusterip my-cs --clusterip="None"`<br>*See* https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/ (last accessed on November 11, 2023). |



## Discovering services

For clients running inside your cluster, Kubernetes supports two primary modes of finding a Service: environment variables and DNS.

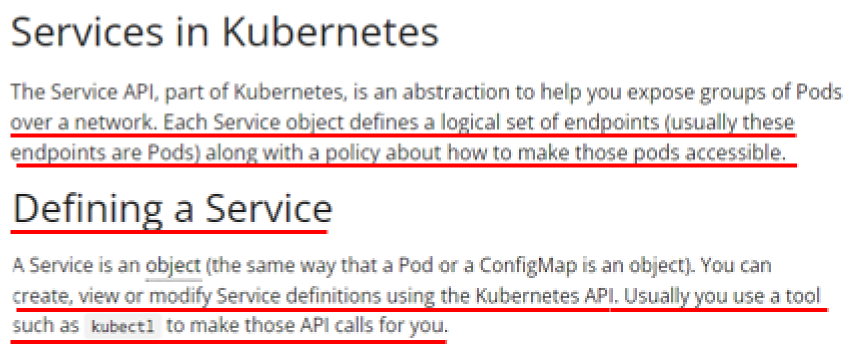
### DNS

You can (and almost always should) set up a DNS service for your Kubernetes cluster using an add-on.

A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name.

For example, if you have a Service called `my-service` in a Kubernetes namespace `my-ns`, the control plane and the DNS Service acting together create a DNS record for `my-service.my-ns`. Pods in the `my-ns` namespace should be able to find the service by doing a name lookup for `my-service` (`my-service.my-ns` would also work).

*See* https://kubernetes.io/docs/concepts/services-networking/service (last accessed on November 11, 2023).



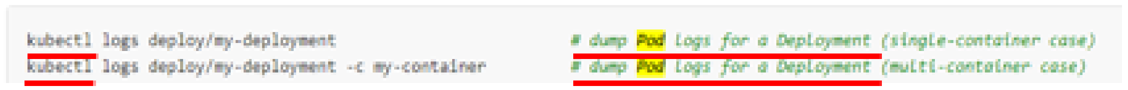
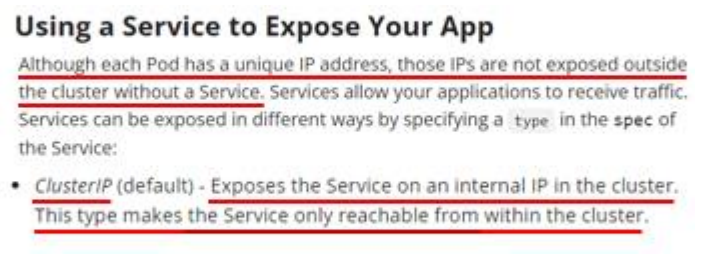
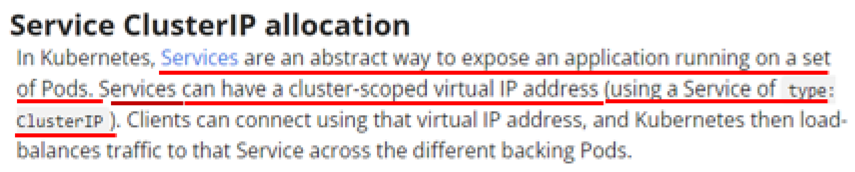
## Services in Kubernetes

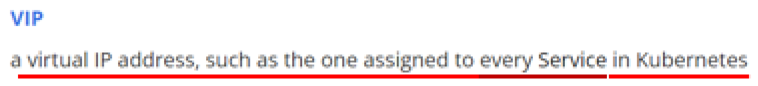
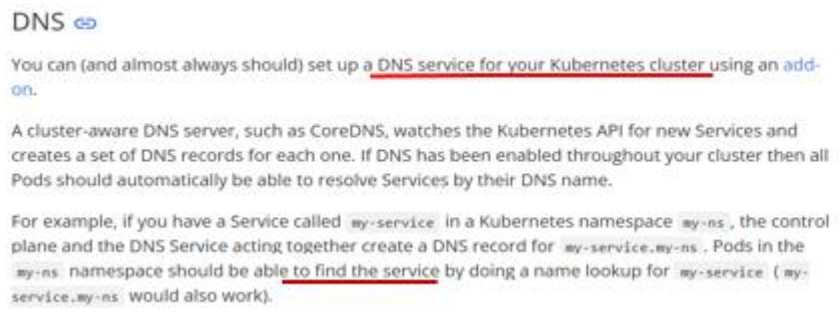
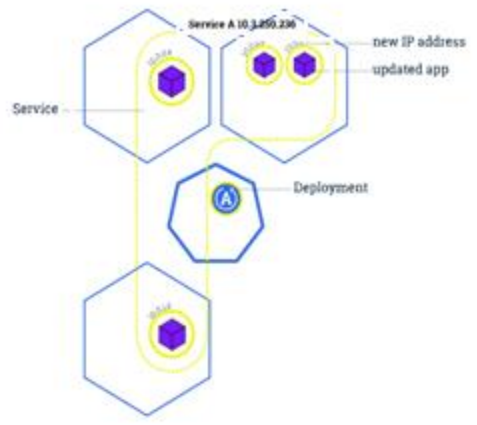
The Service API, part of Kubernetes, is an abstraction to help you expose groups of Pods over a network. Each Service object defines a logical set of endpoints (usually these endpoints are Pods) along with a policy about how to make those pods accessible.

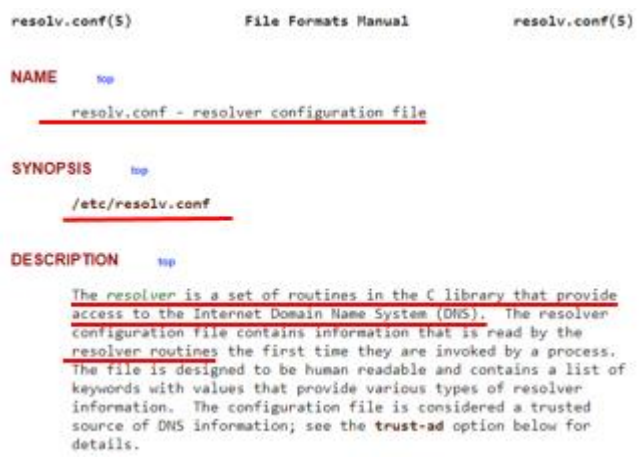
## Defining a Service

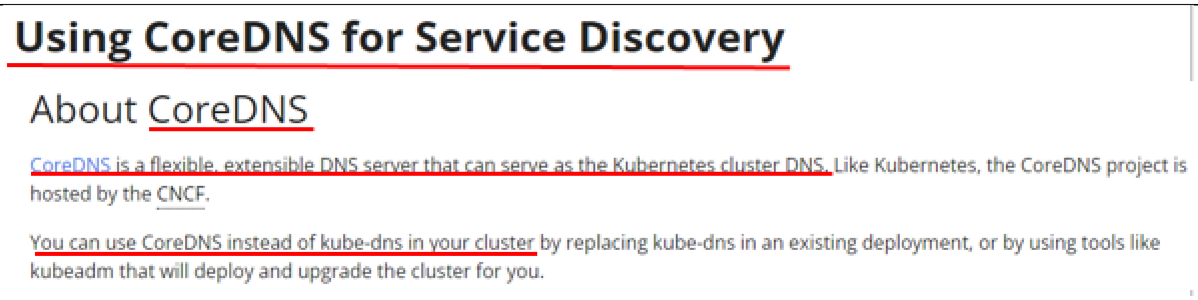
A Service is an object (the same way that a Pod or a ConfigMap is an object). You can create, view or modify Service definitions using the Kubernetes API. Usually you use a tool such as `kubectl` to make those API calls for you.

*See* https://kubernetes.io/docs/concepts/services-networking/service (last accessed on November 11, 2023).

| | |
|---|---|
| |  `kubectl logs deploy/my-deployment                         # dump Pod logs for a Deployment (single-container case)`<br>`kubectl logs deploy/my-deployment -c my-container          # dump Pod logs for a Deployment (multi-container case)`<br><br>*See* https://kubernetes.io/docs/reference/kubectl/cheatsheet/ (last accessed on November 11, 2023). |
| distribute a virtual network address to the device when the device is registered in the virtual network, the device being identified to other devices in the virtual network by the virtual network address; and | Upon information and belief, the Comerica's systems include a register module further configured to "distribute a virtual network address to the device when the device is registered in the virtual network, the device being identified to other devices in the virtual network by the virtual network address." A service runs on a set of pods. A new service is assigned to a cluster IP ("virtual network address"). As part of configuration, the Kubernetes DNS Service module distributes the network address to at least one pod (deice) on which the service runs.<br><br> **Using a Service to Expose Your App**<br>Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a `type` in the `spec` of the Service:<br>• *ClusterIP* (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.<br><br>*See* https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/ (last accessed on November 11, 2023).<br><br> **Service ClusterIP allocation**<br>In Kubernetes, Services are an abstract way to expose an application running on a set of Pods. Services can have a cluster-scoped virtual IP address (using a Service of `type: ClusterIP`). Clients can connect using that virtual IP address, and Kubernetes then load-balances traffic to that Service across the different backing Pods.<br><br>*See* https://kubernetes.io/docs/concepts/services-networking/cluster-ip-allocation/ (last accessed on November 11, 2023). |

| | |
|---|---|
| |  **VIP** a virtual IP address, such as the one assigned to every Service in Kubernetes <br><br> *See* https://kubernetes.io/docs/tutorials/services/source-ip/ (last accessed on November 11, 2023). <br><br>  DNS <br> You can (and almost always should) set up a DNS service for your Kubernetes cluster using an add-on. <br> A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name. <br> For example, if you have a Service called `my-service` in a Kubernetes namespace `my-ns`, the control plane and the DNS Service acting together create a DNS record for `my-service.my-ns`. Pods in the `my-ns` namespace should be able to find the service by doing a name lookup for `my-service` (`my-service.my-ns` would also work). <br><br> *See* https://kubernetes.io/docs/concepts/services-networking/service (last accessed on November 11, 2023). <br><br>   Service A 10.3.250.236<br>new IP address<br>updated app<br>Service<br>A<br>Deployment |

| | |
|---|---|
| | *See* https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/ (last accessed on November 11, 2023). |
| a DNS server for the virtual network, the DNS server configured to receive a DNS request from a first device in the virtual network and return a network address associated with a network route director, | Upon information and belief, the Comerica's systems include "a DNS server for the virtual network, the DNS server configured to receive a DNS request from a first device in the virtual network." A client/ frontend pod ("a first device") sends a DNS query ("DNS Request") using Kubernetes DNS query functionality, such as CoreDNS or kube-dns, which, includes resolver routines associated with the requesting pod. The DNS server (cluster DNS server (e.g. CoreDNS or kube dns)) is configured to return an IP address of cluster DNS server, which is referenced/returned by the process of the kubelet accessing resolv.conf.  resolv.conf(5) File Formats Manual resolv.conf(5) <br> NAME top <br> resolv.conf - resolver configuration file <br> SYNOPSIS top <br> /etc/resolv.conf <br> DESCRIPTION top <br> The resolver is a set of routines in the C library that provide access to the Internet Domain Name System (DNS). The resolver configuration file contains information that is read by the resolver routines the first time they are invoked by a process. The file is designed to be human readable and contains a list of keywords with values that provide various types of resolver information. The configuration file is considered a trusted source of DNS information; see the **trust-ad** option below for details. <br> *See* https://man7.org/linux/man-pages/man5/resolv.conf.5.html (last accessed on November 11, 2023). |



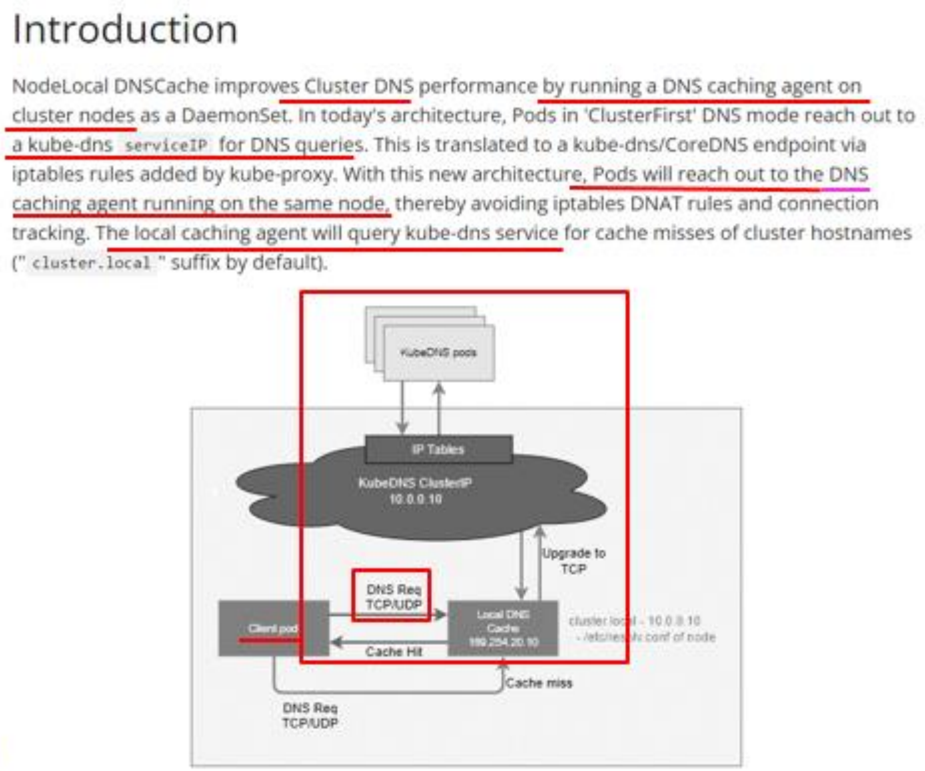
## Using CoreDNS for Service Discovery

### About CoreDNS

CoreDNS is a flexible, extensible DNS server that can serve as the Kubernetes cluster DNS. Like Kubernetes, the CoreDNS project is hosted by the CNCF.

You can use CoreDNS instead of kube-dns in your cluster by replacing kube-dns in an existing deployment, or by using tools like kubeadm that will deploy and upgrade the cluster for you.

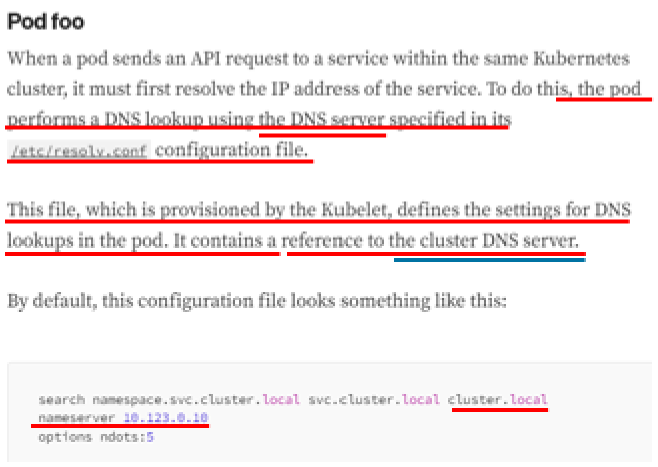
*See* https://kubernetes.io/docs/tasks/administer-cluster/coredns/ (last accessed on November 11, 2023).



### Introduction

NodeLocal DNSCache improves Cluster DNS performance by running a DNS caching agent on cluster nodes as a DaemonSet. In today's architecture, Pods in 'ClusterFirst' DNS mode reach out to a kube-dns `serviceIP` for DNS queries. This is translated to a kube-dns/CoreDNS endpoint via iptables rules added by kube-proxy. With this new architecture, Pods will reach out to the DNS caching agent running on the same node, thereby avoiding iptables DNAT rules and connection tracking. The local caching agent will query kube-dns service for cache misses of cluster hostnames (" `cluster.local` " suffix by default).

*See* https://kubernetes.io/docs/tasks/administer-cluster/nodelocaldns/ (last accessed on November 11, 2023).



**Pod foo**

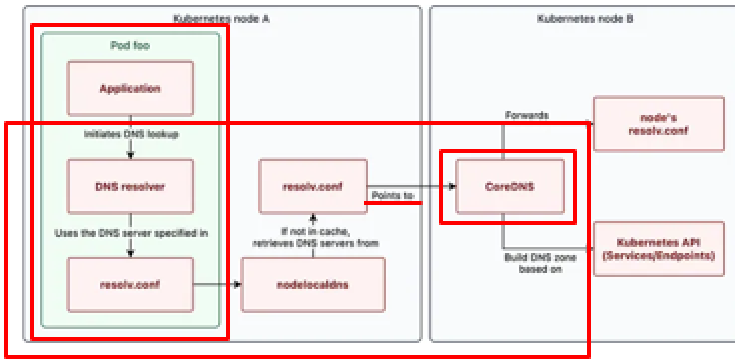
When a pod sends an API request to a service within the same Kubernetes cluster, it must first resolve the IP address of the service. To do this, the pod performs a DNS lookup using the DNS server specified in its `/etc/resolv.conf` configuration file.

This file, which is provisioned by the Kubelet, defines the settings for DNS lookups in the pod. It contains a reference to the cluster DNS server.

By default, this configuration file looks something like this:

```
search namespace.svc.cluster.local svc.cluster.local cluster.local
nameserver 10.123.0.10
options ndots:5
```

## DNS lookups on services



When a pod performs a DNS lookup, the query is first sent to the local DNS resolver in the pod. This resolver uses the `resolv.conf` configuration file. In this file, the `nodelocaldns` server is set up as the default recursive DNS resolver, which acts as a cache.

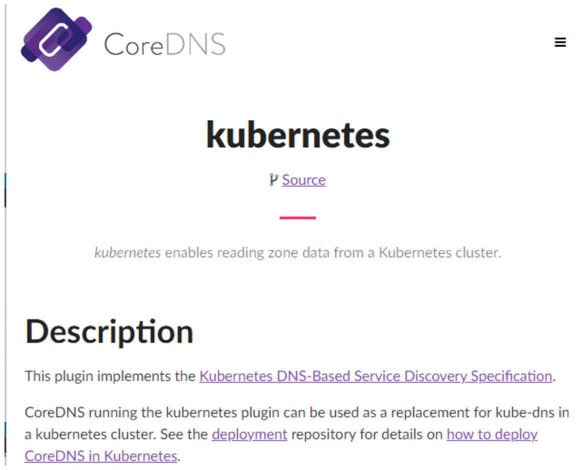
If this cache does not contain the IP address for the requested hostname, the query is forwarded to the cluster DNS server (`CoreDNS`).

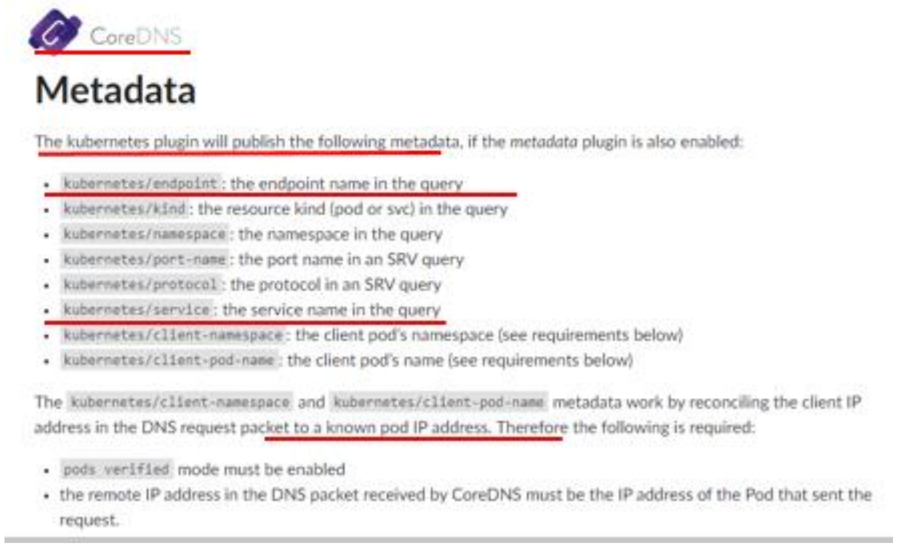
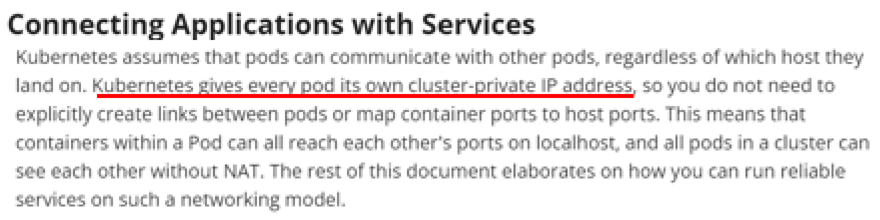
This DNS server determines the IP address by consulting the Kubernetes service registry. This registry contains a mapping of service names to their corresponding IP addresses. This allows the cluster DNS server to return the correct IP address to the requesting pod.

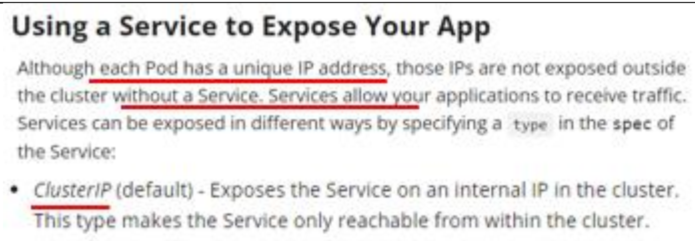
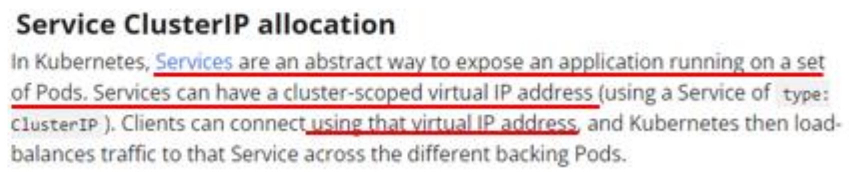
Any domains that are queried but are not in the Kubernetes service registry are forwarded to an upstream DNS server.

We will go through each of these components in more detail step-by-step.

*See* https://medium.com/@seifeddinerajhi/connecting-the-dots-understanding-how-pods-talk-in-kubernetes-networks-992fa69fbbeb (last accessed on November 14, 2023).

| | |
|---|---|
| a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device. | Upon information and belief, the Comerica's systems include "a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device." The private network address associated with a second device is a unique private IP address assigned to a backend pod. The virtual network address associated with the second device is the Cluster IP or Virtual IP assigned to a set of pods.<br><br>CoreDNS<br>kubernetes<br>Source<br>*kubernetes* enables reading zone data from a Kubernetes cluster.<br>**Description**<br>This plugin implements the Kubernetes DNS-Based Service Discovery Specification.<br>CoreDNS running the kubernetes plugin can be used as a replacement for kube-dns in a kubernetes cluster. See the deployment repository for details on how to deploy CoreDNS in Kubernetes. |

| | |
|---|---|
| |  CoreDNS<br>**Metadata**<br>The kubernetes plugin will publish the following metadata, if the *metadata* plugin is also enabled:<br>• `kubernetes/endpoint`: the endpoint name in the query<br>• `kubernetes/kind`: the resource kind (pod or svc) in the query<br>• `kubernetes/namespace`: the namespace in the query<br>• `kubernetes/port-name`: the port name in an SRV query<br>• `kubernetes/protocol`: the protocol in an SRV query<br>• `kubernetes/service`: the service name in the query<br>• `kubernetes/client-namespace`: the client pod's namespace (see requirements below)<br>• `kubernetes/client-pod-name`: the client pod's name (see requirements below)<br>The `kubernetes/client-namespace` and `kubernetes/client-pod-name` metadata work by reconciling the client IP address in the DNS request packet to a known pod IP address. Therefore the following is required:<br>• `pods verified` mode must be enabled<br>• the remote IP address in the DNS packet received by CoreDNS must be the IP address of the Pod that sent the request.<br><br>*See* https://coredns.io/plugins/kubernetes// (last accessed on November 11, 2023).<br><br> **Connecting Applications with Services**<br>Kubernetes assumes that pods can communicate with other pods, regardless of which host they land on. Kubernetes gives every pod its own cluster-private IP address, so you do not need to explicitly create links between pods or map container ports to host ports. This means that containers within a Pod can all reach each other's ports on localhost, and all pods in a cluster can see each other without NAT. The rest of this document elaborates on how you can run reliable services on such a networking model.<br><br>*See* https://kubernetes.io/docs/tutorials/services/connect-applications-service/ (last accessed on November 11, 2023). |

| | |
|---|---|
| | <br>**Using a Service to Expose Your App**<br>Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a `type` in the `spec` of the Service:<br>• *ClusterIP* (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.<br><br>*See* https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/ (last accessed on November 11, 2023).<br><br><br>**Service ClusterIP allocation**<br>In Kubernetes, Services are an abstract way to expose an application running on a set of Pods. Services can have a cluster-scoped virtual IP address (using a Service of `type: ClusterIP` ). Clients can connect using that virtual IP address, and Kubernetes then load-balances traffic to that Service across the different backing Pods.<br><br>*See* https://kubernetes.io/docs/concepts/services-networking/cluster-ip-allocation (last accessed on November 11, 2023). |